CIS 520: Machine Learning *Due:* Apr 13, 2021 (11:59 AM EDT)

Problem Set 4

Submission Instructions:

- **PDF Submission:** This is your main submission, and must contain your solutions to **all problems that you are attempting, including both mathematical problems and programming problems.** It must be submitted as a **s**ingle PDF file to **Gradescope**, compiled in LaTeX using the LaTeX template provided on Canvas. Each problem should be on a new page. Mathematical problems can be either typed in LaTeX or written by hand and **scanned** into images included in your LaTeX solution, but it must be **readable** by our staff to be graded; we recommend that you not take photos of your hand-written solutions. For programming problems, in addition to including any plots, observations, or explanations as requested, be sure to include a snippet of your code in your LaTeX solution; for this, we encourage you to use `lstlisting` environment in `listings` packages for LaTeX.

  Special instruction for submitting on Gradescope: For each problem, select the pages containing your solution for that problem.

- **Code Submission:** Submit all your (Python) code files (**.py files**) to the problem set's code submission component on **Gradescope**. If you use Jupyter notebook, please export them to the corresponding **PS*-P*.py** files and include them in your code submission. Note that these are **in addition** to writing your code inline in the PDF file above. Do not rename any given python files.

- **Python version and libraries:** Make sure that your code runs with **Python 3.8**. Your implementation should **not** use any external libraries other than numpy, scipy and those that are explicitly listed in the handout.

Summary: The PDF file and Python files should be submitted to the corresponding assignments on Gradescope. All components of problem set must be received in the right places and in the right formats before the submission deadline. Plan to submit early!

Collaboration Policy:

You are allowed to discuss problems in groups, but you must write all your solutions and code **individually, on your own**. All collaborators with whom problems were discussed **must** be listed in your PDF submission.

1. (20 points) **Principal Component Analysis.** In this exercise, you are provided part of the MNIST digit data set, containing 3,600 images of handwritten digits (data is provided in `P1/X_train.csv`). Each example is a 28 × 28 grayscale image, leading to 784 pixels. These images correspond to 3 different digits ('3', '6', and '9'), although you do not have the labels for them. You will perform PCA in an unsupervised manner.

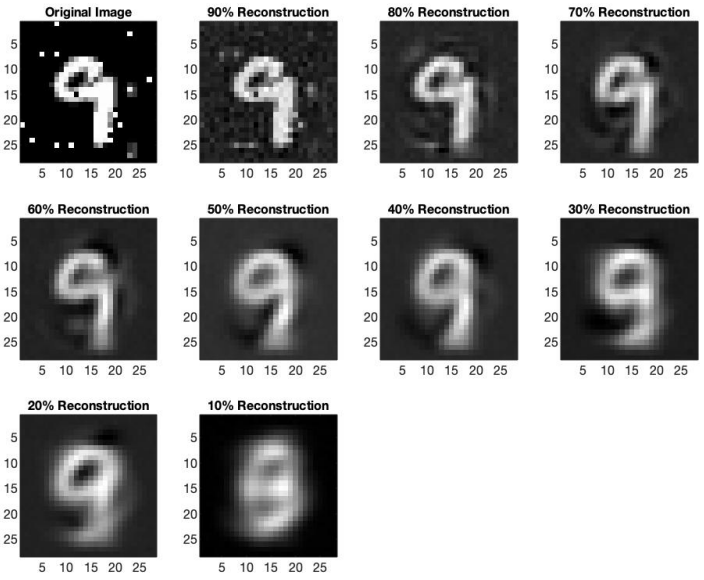
   (a) To get familiar with what the data looks like, generate the last example (the $3,600^{th}$ row of `X_train`) as an image. Provide your code and the resulting image. (*Hint:* You can reshape the data back into a 28 × 28 image using `reshape` in NumPy, and you can use `plt.imshow` with `cmap='gray'` to see the picture. Make sure you are able to get the picture to face the right way, which should look like a "9".)

   (b) Implement PCA using SVD and run it on the data. You can use the function `np.linalg.svd` in NumPy. (*Reminder:* Make sure you understand the NumPy function you used, especially whether the function standardizes the data set before running SVD. Make sure that the data is mean-centered.) Just like in the previous part, generate the first principal component vector as an image. Repeat for the second and third. Do these images make sense? Explain.

   (c) Create a 2D scatter plot showing all the data points projected in the first 2 PC dimensions, clearly labeling your axes. Make sure the data points are mean-centered before projecting onto the principal components. Now create a similar plot showing the same data points projected in the 100^{th} and 101^{st} PC dimensions. What do you observe? Can you explain why you might expect this?

   (d) Graph the (in-sample) fractional reconstruction accuracy as a function of the number of principal components that are included. Also give a table listing the number of principal components needed to achieve each of 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, and 10% reconstruction accuracy (i.e., to explain X% of the variance).

   (e) Using the numbers you found in the previous part, reconstruct the 1000^{th}, 2000^{th}, and 3000^{th} examples using each of the different numbers of principal components. (*Hint:* We have provided a function `plot_reconstruction` for your convenience. Make sure you read the documentation within this function to understand its input arguments.) For instance, the last example looks like:



Python Instructions:

(a) *Please follow the instructions in the comments on input/output specifications. Be aware that the dimension of your vector variables can be a source of bugs.*

(b) *Include all of the codes you used for experiments and plotting in* `PS4-P1.py` *and follow submission instructions when submitting.*

2. (20 points) **EM Practice: Red and Blue Coins.** Your friend has two coins: a red coin and a blue coin, with biases p_r and p_b, respectively (i.e. the red coin comes up heads with probability p_r, and the blue coin does so with probability p_b). She also has an inherent preference π for the red coin. She conducts a sequence of m coin tosses: for each toss, she first picks either the red coin with probability π or the blue coin with probability $1-\pi$, and then tosses the corresponding coin; the process for each toss is carried out independently of all other tosses. You don't know which coin was used on each toss; all you are told are the outcomes of the m tosses (heads or tails). In particular, for each toss i, define a random variable X_i as

$$X_i = \begin{cases} 1 & \text{if the } i\text{-th toss results in heads} \\ 0 & \text{otherwise.} \end{cases}$$

Then the data you see are the values $x_1, \ldots, x_m$ taken by these m random variables. Based on this data, you want to estimate the parameters $\theta = (\pi, p_r, p_b)$. To help with this, for each toss i, define a latent (unobserved) random variable Z_i as follows:

$$Z_i = \begin{cases} 1 & \text{if the } i\text{-th toss used the red coin} \\ 0 & \text{otherwise.} \end{cases}$$

(a) Let X be a random variable denoting the outcome of a coin toss according to the process described above, and let Z be the corresponding latent random variable indicating which coin was used, also as described above (both X and Z take values in $\{0,1\}$ as above). Write an expression for the joint distribution of X and Z. Give your answer in the form

$$p(x, z;\, \theta) = \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\,.$$

(b) Write an expression for the complete-data log-likelihood, $\ln \mathcal{L}_c(\theta) = \sum_{i=1}^{m} \ln p(x_i, z_i;\, \theta)$.

(c) Suppose you knew the values z_i taken by the latent variables Z_i. What would be the maximum-likelihood parameter estimates $\widehat{\theta}$? Give expressions for $\widehat{\pi}$, $\widehat{p}_r$, and $\widehat{p}_b$ (in terms of x_i and z_i). Show your calculations.

(d) In the absence of knowledge of z_i, one possibility for estimating θ is to use the EM algorithm. Recall that the algorithm starts with some initial parameter estimates θ^0, and then on each iteration t, performs an E-step followed by an M-step. Let θ^t denote the parameter estimates at the start of iteration t. In the E-step, for each toss i, the algorithm requires computing the posterior distribution of the latent variable Z_i under the current parameters θ^t. Calculate the posterior probability $\mathbf{P}(Z_i = 1 \,|\, X_i = x_i;\, \theta^t)$.

(e) For each toss i, denote the posterior probability computed in part (d) above by γ_i^t (so that $\gamma_i^t = \mathbf{P}(Z_i = 1 \,|\, X_i = x_i;\, \theta^t)$). Then the expected complete-data log-likelihood with respect to these posterior distributions is

$$\sum_{i=1}^{m} \Big(\gamma_i^t \cdot \ln p(x_i, 1;\, \theta) + (1 - \gamma_i^t) \cdot \ln p(x_i, 0;\, \theta) \Big)\,.$$

The M-step of the EM algorithm requires finding parameters θ^{t+1} that maximize this expected complete-data log-likelihood. Determine the updated parameters θ^{t+1}. Give expressions for π^{t+1}, p_r^{t+1}, and p_b^{t+1} (in terms of x_i and γ_i^t). Show your calculations.

3. (25 points) **Programming Exercise: Gaussian Mixture Models.** Write a piece of Python code to implement the EM algorithm for learning a Gaussian mixture model (GMM) given a data set $\mathbf{X}$, where $\mathbf{X}$ is an $m \times d$ matrix (m instances, each of dimension d), and a target number of Gaussians K: your program should take the training data $\mathbf{X}$ and the target number of Gaussians K as input and should output estimated parameters of a mixture of K Gaussians (specifically, it should output mixing coefficients $\widehat{\boldsymbol{\pi}} \in \Delta_K$, mean vectors $\widehat{\boldsymbol{\mu}}_1, \ldots, \widehat{\boldsymbol{\mu}}_K \in \mathbb{R}^d$, and covariance matrices $\widehat{\boldsymbol{\Sigma}}_1, \ldots, \widehat{\boldsymbol{\Sigma}}_K \in \mathbb{R}^{d \times d}$). For this problem, you are provided a 2-dimensional data set generated from a mixture of (3) Gaussians. The data set is divided into training and test sets; the training set has 480 data points and the test set has 120 data points. The goal is to learn a GMM from the training data; the quality of the learned GMM will be measured via its log-likelihood on the test data.

EM initialization and stopping criterion: Initialize the mixing coefficients to a uniform distribution, and each of the covariance matrices to be the $d \times d$ identity matrix: $\boldsymbol{\pi}^0 = (1/K, \ldots, 1/K)^\top$, and $\boldsymbol{\Sigma}_1^0 = \ldots = \boldsymbol{\Sigma}_K^0 = \mathbf{I}_d$. Initializations for the means will be provided to you (if you like, you can write your program to take the mean initialization as an additional input). For the stopping criterion, on each iteration t, keep track of the (incomplete-data) log-likelihood on the training data, $\sum_{i=1}^m \ln p(x_i;\, \theta^t)$; stop when either the change in log-likelihood, $\sum_{i=1}^m \ln p(x_i;\, \theta^{t+1}) - \sum_{i=1}^m \ln p(x_i;\, \theta^t)$, becomes smaller than 10^{-6}, or when the number of iterations reaches 1000 (whichever occurs earlier; here θ^t denotes the parameter estimates on iteration t).

   (a) **Known K.** For this part, assume you are given $K = 3$.

      i. **Learning curve.** Use your implementation of EM for GMMs to learn a mixture of 3 Gaussians from increasing fractions of the training data (10% of the training data, then 20% of the training data, then 30% and so on upto 100%). You must use the subsets provided in the folder `P3/TrainSubsets`; in each case, to initialize the means of the Gaussians, use the initializations provided in the folder `P3/MeanInitialization`. In each case, calculate the **normalized** (incomplete-data) log-likelihood of the learned model on the training data, as well as the normalized log-likelihood on the test data (here normalizing means dividing the log-likelihood by the number of data points). In a single plot, give curves showing the normalized train and test log-likelihoods (on the y-axis) as a function of the fraction of data used for training (on the x-axis). (Note: the training log-likelihood should be calculated only on the subset of examples used for training, not on all the training examples available in the given data set.)

      ii. **Analysis of learned models.** For each of the learned models (corresponding to the different subsets of the training data), show a plot of the Gaussians in the learned GMM overlaid on the test data. What do you observe? For the final model (learned from 100% of the training data), also write down all the parameters of the learned GMM, together with the (normalized) train and test log-likelihoods.

   Hints:

      i. *You may use the function we provide in* `plot_multiple_contour_plots` *to assist in plotting the density curves. Be sure to carefully read the function header, so you know how to structure your inputs.*

   (b) **Unknown K.** Now suppose you do not know the right number of Gaussians in the mixture model to be learned. Select the number of Gaussians K from the range $\{1, \ldots, 5\}$ using 5-fold cross-validation on the full training data (use the folds provided in the folder `P3/CrossValidation`). For each K, to initialize the means of the K Gaussians, use the initializations provided in the folder `P3/MeanInitialization`. In a single plot, draw 3 curves showing the (normalized) train, test, and cross-validation log-likelihoods (on the y-axis) as a function of number of Gaussians K (on the x-axis); Write down the chosen value of K (with the highest cross-validation log-likelihood).

   Python Instructions:

(a) *Please follow the instructions in the comments on input/output specifications. Be aware that the dimension of your vector variables can be a source of bugs.*

(b) *Implement the* `E_step`*,* `M_step`*, and* `compute_llh` *methods in* `gmm.py`*. You may use the function* `scipy.stats.multivariate_normal` *in your implementation.*

(c) *Include all of the codes you used for experiments and plotting in* `PS4-P3.py` *and follow submission instructions when submitting.*

4. (35 points) **Hidden Markov Models.** On any given day, Alice is in one of two states: happy or sad. You do not know her internal state, but get to observe her activities in the evening. Each evening, she either sings, goes for a walk, or watches TV.

Alice's state on any day is random. Her state Z_1 on day 1 is equally likely to be happy or sad:

$$P(Z_1 = \mathsf{happy}) = 1/2\,.$$

Given her state Z_t on day t, her state Z_{t+1} on the next day is governed by the following probabilities (and is conditionally independent of her previous states and activities):

$$P(Z_{t+1} = \mathsf{happy} \,|\, Z_t = \mathsf{happy}) = 4/5 \qquad P(Z_{t+1} = \mathsf{happy} \,|\, Z_t = \mathsf{sad}) = 1/2\,.$$

Alice's activities are also random. Her activities vary based on her state; given her state Z_t on day t, her activity X_t on that day is governed by the following probabilities (and is conditionally independent of everything else):

$$\begin{array}{ll} P(X_t = \mathsf{sing} \,|\, Z_t = \mathsf{happy}) = 5/10 & P(X_t = \mathsf{sing} \,|\, Z_t = \mathsf{sad}) = 1/10 \\ P(X_t = \mathsf{walk} \,|\, Z_t = \mathsf{happy}) = 3/10 & P(X_t = \mathsf{walk} \,|\, Z_t = \mathsf{sad}) = 2/10 \\ P(X_t = \mathsf{TV} \,|\, Z_t = \mathsf{happy}) = 2/10 & P(X_t = \mathsf{TV} \,|\, Z_t = \mathsf{sad}) = 7/10\,. \end{array}$$

(a) (15 points) Suppose you observe Alice singing on day 1 and watching TV on day 2, i.e. you observe $x_{1:2} = (\mathsf{sing}, \mathsf{TV})$. Find the joint probability of this observation sequence together with each possible hidden state sequence that could be associated with it, i.e. find the four probabilities below. Show your calculations.

$$\begin{array}{l} P\big(X_{1:2} = (\mathsf{sing}, \mathsf{TV}), Z_{1:2} = (\mathsf{happy}, \mathsf{happy})\big) \\ P\big(X_{1:2} = (\mathsf{sing}, \mathsf{TV}), Z_{1:2} = (\mathsf{happy}, \mathsf{sad})\big) \\ P\big(X_{1:2} = (\mathsf{sing}, \mathsf{TV}), Z_{1:2} = (\mathsf{sad}, \mathsf{happy})\big) \\ P\big(X_{1:2} = (\mathsf{sing}, \mathsf{TV}), Z_{1:2} = (\mathsf{sad}, \mathsf{sad})\big) \end{array}$$

Based on these probabilities, what is the most likely hidden state sequence $z_{1:2}$? What is the individually most likely hidden state on day 2 ?

(b) (20 points) Write a small piece of Python code to implement the Viterbi algorithm, and use this to calculate both the most likely hidden state sequence $z_{1:10}$ and the corresponding maximal joint probability $p(x_{1:10}, z_{1:10})$ for each of the following observation sequences:

$$\begin{array}{l} x_{1:10} = (\mathsf{sing}, \mathsf{walk}, \mathsf{TV}, \mathsf{sing}, \mathsf{walk}, \mathsf{TV}, \mathsf{sing}, \mathsf{walk}, \mathsf{TV}, \mathsf{sing}) \\ x_{1:10} = (\mathsf{sing}, \mathsf{sing}, \mathsf{sing}, \mathsf{TV}, \mathsf{TV}, \mathsf{TV}, \mathsf{TV}, \mathsf{TV}, \mathsf{TV}, \mathsf{TV}) \\ x_{1:10} = (\mathsf{TV}, \mathsf{walk}, \mathsf{TV}, \mathsf{sing}, \mathsf{sing}, \mathsf{sing}, \mathsf{walk}, \mathsf{TV}, \mathsf{sing}, \mathsf{sing}) \end{array}$$

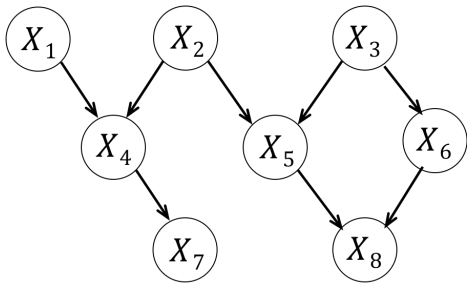
In your code, rather than multiplying probabilities (which can lead to underflow), you may find it helpful to add their logarithms (and later exponentiate to obtain the final result). Include a snippet of your code in your LaTeX submission.

Python Instructions:

(a) *Please follow the instructions in the comments on input/output specifications. Be aware that the dimension of your vector variables can be a source of bugs.*

(b) *Include your implementation of the Viterbi algorithm* `viterbi` *and all of the codes you used for experiments and plotting in* `PS4-P4.py` *and follow submission instructions when submitting.*

5. (20 points) **Bayesian Networks.** Consider the Bayesian network over 8 random variables X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8 shown below (assume for simplicity that each random variable takes 2 possible values):



(a) Write an expression for the joint probability mass function $p(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ that makes the same (conditional) independence assumptions as the Bayesian network above.

(b) Consider a joint probability distribution satisfying the following factorization:

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = p(x_1)p(x_2)p(x_3)p(x_4 \mid x_1)p(x_5 \mid x_2, x_3)p(x_6 \mid x_3, x_5)p(x_7)p(x_8 \mid x_5)\,.$$

Is this distribution included in the class of joint probability distributions that can be represented by the Bayesian network above? Briefly explain your answer.

(c) If the edge from X_5 to X_8 is removed from the above network, will the class of joint probability distributions that can be represented by the resulting Bayesian network be smaller or larger than that associated with the original network? Briefly explain your answer.

(d) Given the above figure, determine whether each of the following is true or false. Briefly justify your answer.

i. $X_4 \perp X_5$
ii. $X_2 \perp X_6$
iii. $X_3 \perp X_4 \mid X_8$
iv. $X_2 \perp X_8 \mid X_5$